An Empirical Study of Tokenization Strategies for Biomedical Information Retrieval

Jing Jiang
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
jiang4@cs.uiuc.edu

ChengXiang Zhai
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
czhai@cs.uiuc.edu

ABSTRACT

Due to the great variation of biological names in biomedical text, appropriate tokenization is an important preprocessing step for biomedical IR. Despite its importance, there has been little study on the evaluation of various tokenization strategies for biomedical text. In this work, we conducted a careful, systematic evaluation of a set of tokenization heuristics on all the available TREC biomedical IR test collections with two representative retrieval methods. We also studied the effect of stemming and stop word removal on the retrieval performance. As expected, our experimental results show that tokenization can significantly affect the retrieval accuracy; appropriate tokenization can improve the performance by up to 80%. In particular, it is shown that different query types require different tokenization heuristics, stemming is effective only for certain queries, and stop word removal in general does not improve the retrieval performance in biomedical text.

1. INTRODUCTION

Recently, the growing amount of scientific literature in genomics and related biomedical disciplines has led to an increasing amount of interest in and need for applying IR as well as other text management techniques to access the biomedical textual data. The special language usage in biomedical literature, such as the frequent occurrences of gene symbols and the use of inconsistent lexical variants of the same genes, has raised many new challenges in the biomedical IR field.

In previous work on biomedical IR, while many efforts have been put to query expansion and synonym normalization, little attention has been paid to tokenization and other text preprocessing steps that transform the documents and the queries into the bag-of-word representation. Although tokenization is not a critical step for retrieval in English text in general domains, it is not a trivial task for languages in special domains (sometimes referred to as sublanguages), largely due to the domain-specific terminologies.

IR methods generally rely on term matching. The purpose of tokenization is therefore to break down the text into *tokens* (or *terms*), which are small units of meaningful text, such that a match between a token in the query and a token in a document can in general increase our confidence that the document is relevant to the query. It is often also desirable during the preprocessing stage to normalize tokens that look similar and convey the same meaning into a single canonical form. For English text in general domains, individual English words are naturally used as tokens. Tokenization can be done by simply using white spaces as delimiters, or in a slightly more sophisticated way, by using all non-alphanumerical characters as delimiters. Stemming is often used to normalize the morphological variants of the same base word. In biomedical text, however, the content words include not only English words, but also many special terms such as the names of genes, proteins and chemicals. These names often contain special characters such as numerals, hyphens, slashes and brackets, and the same entity often has different lexical variants. Clearly, a simple tokenizer for general English text cannot work well in biomedical text. If all non-alphanumerical characters inside a named entity are used as delimiters to separate the name into several tokens, the proximity of these tokens is lost in the bag-of-word representation, which may result in a loss of the semantic meaning of the tokens and cause mismatches. Moreover, breaking named entities into fragments may affect the *tf·idf* weighting of the tokens in an unwanted way. On the other hand, if all the non-alphanumerical characters are kept, it is hard to capture minor variations of the same name.

Table 1 shows an example taken from Topic 38 of the ad hoc retrieval task in TREC 2003 Genomics Track, where two tokenizers produced different matching results. The original query contains the gene symbol *MIP-1-alpha*. There are three lexical variants of this symbol that appear in the judged relevant documents, as shown in the 2nd column of the 3rd, 4th and 5th rows of Table 1. The 2nd column of the 6th row shows a piece of text that can possibly become a mismatch. Tokenizer 1 uses all non-alphanumerical characters as token delimiters. Tokenizer 2 removes special characters such as hyphens and brackets. The 3rd and the 5th columns of Table 1 show the tokenized text. We can see that Tokenizer 1 captures one lexical variant, but misses the other two variants and generates one mismatch. Tokenizer 2, however, captures two variants and ignores the mismatch. Thus Tokenizer 2 is superior to Tokenizer 1 in this particular case. Because many queries in biomedical IR contain gene names and symbols like the one in the above example, it is important to find a suitable tokenization strategy in order to generate the best retrieval results.

Despite its importance, to the best of our knowledge, there has not been any work devoted to systematic comparison of different tokenization strategies for biomedical IR. In this paper, we present a set of tokenization heuristics that are

Table 1: An Example of The Effect of Two Tokenization Strategies

Variant	Original Text	Tokenizer 1	Match?	Tokenizer 2	Match?
In Query	*MIP-1-alpha*	mip 1 alpha	N/A	mip1alpha	N/A
Var. 1	*MIP-1alpha*	mip 1alpha	No	mip1alpha	Yes
Var. 2	*(MIP)-1alpha*	mip 1alpha	No	mip1alpha	Yes
Var. 3	*MIP-1 alpha*	mip 1 alpha	Yes	mip1 alpha	No
Mismatch	*MIP-1 beta*, *IFN-alpha*	mip 1 beta ifn alpha	Yes	mip1beta ifnalpha	No

generalized from previous work on biomedical IR, and we conduct a systematic evaluation of these heuristics. In particular, we define three sets of *break points*, three break point normalization methods, and a Greek alphabet normalization method. We also study the effect of stemming and stop word removal for biomedical text as these preprocessing steps may also affect the retrieval performance. The goal of our study is to provide a set of standard tokenization heuristics that are in general suitable for biomedical IR.

We evaluate the tokenization heuristics on the data from TREC 2003, 2004 and 2005's Genomics Track, which includes all available TREC biomedical IR test collections. We use two representative retrieval methods. Results from both retrieval methods show that tokenization strategies can affect the retrieval performance significantly; good tokenization can improve the performance by up to 80%. For different types of queries, different sets of tokenization heuristics should be applied in order to achieve the optimal performance. In particular, for queries with only gene symbols, removing a set of special characters inside the tokens and replacing Greek letters with Latin letters are shown to be effective. In contrast, for queries with only full gene names and for verbose queries that also contain English words to describe the information need, replacing special characters with spaces produces the best results. In addition, for verbose queries, stemming further improves the performance. Stop word removal does not help retrieval in general, except for verbose queries containing common English words, where removing stop words may slightly improve the performance.

The rest of the paper is organized as follows: In Section 2, we survey the tokenization strategies explored in previous work on biomedical IR. In Section 3 and Section 4, we generalize the various tokenization strategies into a set of heuristics, and explain the rationale behind each heuristic. We evaluate the different tokenization heuristics in Section 5. In Section 6, we conclude our work by recommending a set of suitable tokenization heuristics based on the query types.

2. RELATED WORK

Most previous work on biomedical IR appeared in the Genomics Track in TREC 2003, 2004 and 2005. To address the prevalent name variation problem in biomedical text, most groups focused on query expansion, either by using external knowledge bases such as LocusLink to find gene synonyms [2, 6], or by generating lexical variants of the gene names in the queries using heuristic rules [2, 8]. Tokenization of the corpus was not seriously studied, and most groups did not describe their tokenization strategies in detail.

Among the work that mentioned special tokenization techniques, [14] pointed out that allowing a token to contain both alphabetical and numerical characters was a little better than separating them. In contrast, [10, 4] chose to separate alphabetical and numerical strings in order to handle hyphenation of the alphabetical and the numerical parts in a gene name. But [10] also imposed proximity search to ensure that the separated components were close together in the retrieved documents. [5] allowed the following special characters to be part of a token provided that they were not the first or the last character of the token: (,), [,], ', -, ',' and /. Thus names such as *1,25-dihydroxyvitamin* and *dead/h* would become single tokens.

Some work also considered combining adjacent words that were separated by spaces into single terms. [13] used a simple rule to combine a short-length word and its adjacent non-short-length word into a single keyword. They reported a 15% improvement over the baseline using this simple heuristic. [1] combined adjacent alphabetical chunks and numerical chunks into token bigrams.

In contrast to the little amount of report on the tokenization strategies applied to the document collection, there has been much more report on different ways to tokenize the queries and to generate alternative gene names using heuristic rules. [8] defined *break-points* in tokens where hyphens and spaces can be inserted or removed without changing the meaning of the token. [2, 8] also considered replacing Greek letters with their Latin equivalents to generate lexical variants of gene names in the queries.

Although Porter stemmer was the most commonly used English stemmer, a few groups reported experiments with different stemmers on biomedical text. [13] showed that Porter stemmer could decrease the performance while Lovins stemmer improved the performance. [12] showed that S stemmer sometimes was advantageous over Lovins stemmer. Many groups also removed stop words from the document collection using an external stop word list [13, 3, 6]. A commonly used stop word list for biomedical text is the one from PubMed [3]. There has not been any work comparing the retrieval performance with and without stop word removal.

3. TOKENIZATION HEURISTICS

In this section, we generalize the various tokenization strategies in previous work into a set of organized heuristics. Such generalization allows us to better understand the rationale behind each strategy, and to systematically evaluate them.

For all the tokenization strategies we consider, we assume that the last step of tokenization is to change all upper case letters into lower cases. This case normalization is done in the very end because some tokenization heuristics rely on

Table 2: Heuristic Rules to Remove Non-Functional Characters

1) replace the following characters with spaces: ! " # $ % & * < = > ? @ \ | ~
2) remove the following characters if they are followed by a space: . : ; ,
3) remove the following pairs of brackets if the open bracket is preceded by a space and the close bracket is followed by a space: ( ) [ ]
4) remove the single quotation mark if it is preceded by a space or if it is followed by a space: '
5) remove "'s" and "'t" if they are followed by a space
6) remove slash "/" if it is followed by a space

the cases of letters to determine where to break the text. Also, the tokenization strategies that we consider do not include the ones that combine adjacent words originally separated by spaces into bigram tokens. We do not include such strategies in our evaluation because of two reasons: (1) The candidate tokens to form bigrams are alphabetical strings followed by numerical strings or vice versa, but numerical strings appear very frequently in biomedical text, and most of them are not part of an entity name. (2) Because we can never be sure that the combined bigrams can replace the original unigram tokens, we need to keep the original unigram tokens in the documents. But such arbitrary expansion of the documents may cause inaccurate estimation of various numbers that are used in retrieval formulas, such as the document length in the *tf·idf* method and the term probability in the language modeling approach.

Before we go into the various tokenization heuristics, we first define a naive strategy as the very basic baseline to compare against. This naive method uses only white spaces as the delimiters to break down the text into tokens.

3.1 Removal of Non-Functional Characters

In biomedical text, although non-alphanumerical characters are frequently used to help represent various kinds of entities and other biomedical information, the most important special characters that make retrieval difficult are those that frequently occur in gene names and protein names, such as hyphens, slashes, and brackets. Many other non-alphanumerical characters such as '=' and '#' usually do not occur inside an entity name or convey any important semantic meaning. These "non-functional" characters can thus be excluded from the tokens. Previous work that had special handling of non-alphanumerical characters also only focused on a subset of non-alphanumerical characters. Therefore, as a first step, we manually identified a set of special characters that we believe can be safely discarded, and we defined a set of rules in the form of regular expressions to remove these special characters. Table 2 lists the heuristic rules we defined to remove such non-functional characters. How much the retrieval performance is affected by removing these non-functional characters will be determined by empirical experimental results.

3.2 Break Points

After the non-functional characters are removed, the remaining text consists of alphanumerical characters and a set of special characters as listed in Table 3. The special characters in set 1 are often used to separate the several components of an entity name, such as in *(MIP)-1alpha*, *pRB/p105*, and *TrpEb_1*. The special characters in set 2 are not very frequently used for gene names, but rather mostly used inside numbers like *0.20* and *20,000*, inside chemical formulas like the ions *Ca2+* and *Na+*, or inside names of chemical compounds and DNA sequences to describe the structure of those entities, like in *1,2,3,4-TeCDD*, *2,3,7,8-TeCDD* and *2',5'-linked 3'-deoxyribonucleotides*. In most of these cases, it is not necessary to divide the two alphanumerical strings around those special characters in set 2 into different components.

Table 3: Special Characters Left in the Text after Removal of Non-Functional Characters

Special Character Set 1	Special Character Set 2
( ) [ ] - _ /	. : ; , ' +

Besides these special characters listed in Table 3, there are also other places within strings where we should consider breaking the strings into smaller tokens. These are the places where an alphabetical character changes to a numerical character and vice versa, or where a sequence of upper case letters changes to a sequence of lower case letters and vice versa. Formally, following the work in [8], we use the following rules to define three kinds of hidden places where strings can be further broken down: (1) between an alphabetical character on the left(right) and a numerical character on the right(left), (2) between a lower case letter on the left and an upper case letter on the right, and (3) between an upper case letter on the left and a lower case letter on the right, unless the upper case letter is preceded by a space, or by a numerical character or another lower case letter (in which case the upper case letter will be separated from its previous lower case letter by rule (2)). With these rules, gene symbols such as *MIP-1alpha* and *MIP-1-alpha* can both be broken down into *MIP 1 alpha*, making it possible to match them.

Borrowing the term from [8], we refer to all the special characters listed in Table 3 and the three kinds of hidden places defined above as *break points*. The break points are the places where an entity name can be potentially broken down into smaller components such that if we connect these smaller components differently, we could form a lexical variant of the original name. Because of the different degrees to which we believe these break points should be used, we consider three sets of break points. Break Point Set 1 (BP1) consists of the special characters in Special Character Set 1 in Table 3, Break Point Set 2 (BP2) consists of both Special Character Set 1 and Special Character Set 2 in Table 3, and Break Point Set 3 (BP3) consists of all special characters in BP2 and the hidden break points defined by the three rules.

3.3 Break Point Normalization

With the break points we defined in the last section, we can now normalize the different lexical variants of the same entity by normalizing the break points into the same representation. There are different ways to normalize the break points. One way is to replace all the break points with a sin-

gle special character, such as a hyphen, and keep these break points designated by hyphens in the final tokens. Thus, if we use Break Point Set 3, gene symbols *MIP-1-alpha*, *MIP-1alpha* and *(MIP)-1alpha* will all become the same single token *MIP-1-alpha*. Another way to normalize the break points is to replace all of them with spaces. Thus, using BP3, gene symbols *MIP-1-alpha*, *MIP-1alpha* and *(MIP)-1alpha* will all become three tokens: *MIP*, *1*, and *alpha*. There are advantages and disadvantages of both normalization methods. For the first one, the proximity of the components of a gene name is preserved, ensuring high precision in matching entity names. However, it could not handle the case when one lexical variant contains a space while another lexical variant has a break point in the place of the space, such as in *MIP-1 alpha* and *MIP-1-alpha*. The second normalization method can handle this case well because all break points are replaced with spaces. However, proximity of the components of the name is lost, which may cause mismatches.

There is another problem with the two normalization methods described above. Sometimes a hidden break point cannot be captured by the three rules we defined in Section 3.2. For example, the topics in TREC 2003 Genomics Track contain these gene alias symbols: *Pkca* and *Prkca* (for the gene "*Protein kinase C, alpha*"), *Tcra* and *Tcralpha* (for the gene "*T-cell receptor alpha*"), and *Ifnb2* (for the gene "*Interleukin 6 (interferon, beta 2)*"). We can see that the Greek letters *alpha* and *beta*, or their Latin equivalents *a* and *b*, cannot be clearly distinguished from the rest of the text in these symbols. Thus if a hyphen is inserted into such a hidden break point, this hyphenated variant cannot be matched with the one without the hyphen by either of the normalization methods we described above. Because such hidden break points are too hard to detect by any simple regular expressions, one way to solve the problem is to normalize all variants into the form without hyphens or other special characters. We have not seen such an approach in any previous work, but we think this approach is a reasonable solution to the problem with undetectable break points.

To summarize, we consider three methods to normalize the break points. Method one replaces all break points with hyphens (or inserts hyphens into hidden break points). We call this method the *Hyphen-Normalization* method, or *H-Norm*. Method two replaces all break points with spaces (or inserts spaces into hidden break points). We call this method the *Space-Normalization* method, or *S-Norm*. Method three removes all break points (or does nothing to hidden break points). We call this method the *Join-Normalization* method, or *J-Norm*. Note that all three normalization methods can be used in conjunction with any of the three sets of break points, except that when J-Norm is used, BP3 becomes essentially the same as BP2.

3.4 Greek Alphabet Normalization

Another heuristic that has been previously explored is to replace Greek letters with their Latin equivalents. In biomedical text, entity names often contain Greek letters such as *alpha*, *beta*, etc. Sometimes these Greek letters are abbreviated as *a*, *b*, etc., but there is no consistent rules as to when the Greek letters should be abbreviated. A simple method to tackle this problem is to replace all occurrences of Greek letters with the Latin letters that are equivalent to them. Note that sometimes a Greek letter can be embedded in an alphabetical string and hard to detect, such as in *Tcralpha*. We do not try to replace these Greek letters as there is no easy way to detect these hidden occurrences of Greek letters. Thus, our replacement rule is to check each maximum span of consecutive alphabetical letters in the text, and replace the ones that are in the Greek alphabet. We call this Greek letter replacement strategy the *Greek-Normalization* heuristic, or *G-Norm*. Note that while the three normalization methods describe in Section 3.3 are mutually exclusive, *G-Norm* is orthogonal to those three normalization methods, and thus can be applied on top of any of them.

4. STEMMING AND STOP WORD REMOVAL

After tokenization, stemming is an optional step to further normalize the tokens. Based on previous work that explored stemming algorithms for biomedical IR, we consider three stemmers in our evaluation: the Porter stemmer [11], the Lovins stemmer [9], and the S stemmer [7]. S stemmer only removes a few common word endings. Lovins stemmer is more aggressive than Porter stemmer, which in turn is more aggressive than S stemmer.

We also consider two stop word removal methods. One method uses an external stop word list. In our experiments, we use the stop word list from PubMed. Since stop words are essentially the most frequent words in a document collection, the second method we consider uses a stop word list generated from the document collection itself by extracting the most frequent k tokens.

5. EVALUATION

In this section, we show our empirical evaluation of the set of tokenization heuristics we described in Section 3 and Section 4. Specifically, our goal is as follows. For removal of the non-functional special characters, intuitively it should improve the performance, because most of the noise caused by punctuation such as periods and commas is removed by this heuristic. The focus of the evaluation of this heuristic is thus to see whether this non-functional character removal step is safe for most of the queries. For the three sets of break points we defined, BP1, BP2, and BP3, the goal of the evaluation is to see which set gives the best retrieval performance when it is used in conjunction with break point normalization. Similarly, for the three break point normalization methods, the goal is the find the best normalization method for retrieval. For Greek alphabet normalization, we want to see whether this replacement can improve the retrieval performance. Lastly, for stemming and stop word removal, we want to see whether stemming improves the performance and which stemmer performs the best, and wether removing stop words improves the performance.

We also need to make our evaluation of tokenization strategies independent of the retrieval method being used so that the best tokenization strategies we find can be used for any standard IR method. We thus choose two representative retrieval methods to use in our evaluation: a *tf·idf* retrieval method with BM25 term frequency weighting, and the KL-divergence retrieval method [15], which represents the language modeling approach. The parameters for both methods are tuned in each experiment for each set of queries because the parameters are sensitive to the query type and

Table 4: Comparison between Naive Method and Baseline Method

	Keyword Queries				Verbose Queries					
	03 Symbol		03 Name		04		05 Gene		05 Non-gene	
	KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF
Naive	0.1451	0.1546	0.0833	0.0919	0.1672	0.1736	0.1921	0.1929	0.1302	0.1166
Baseline	0.1520	0.1643	0.0892	0.0971	0.2694	0.2700	0.2296	0.2189	0.1549	0.1413
% Impr.	**4.76%**	**6.27%**	**7.08%**	**5.66%**	**61.1%**	**55.5%**	**19.5%**	**13.5%**	**19.0%**	**21.2%**
# Decr.	23/50	22/50	13/50	15/50	6/50	5/50	10/39	12/39	4/10	4/10

to the tokenization heuristics used. We use the Lemur Language Modeling Toolkit[1] for our experiments.

In all our experiments, we use the average MAP measure over the set of queries as the evaluation metric.

5.1 Document Collections and Queries

The document collections and the queries we use for evaluation are from the ad hoc retrieval task in TREC 2003, 2004 and 2005's Genomics Track[2]. The document collection used in 2003 TREC Genomics Track contains 525,938 MEDLINE records between April 2002 and April 2003. The collection used in 2004 and 2005 is a 10-year subset of MEDLINE records from 1994 to 2003.

The topics used in the three years' Genomics Track represent different types of queries and different information need. The 50 topics from TREC 2003 each consist of a gene and an organism name with the specific retrieval task formally stated as follows: For gene X, find all MEDLINE references that focus on the basic biology of the gene or its protein products from the designated organism. Basic biology includes isolation, structure, genetics and function of genes/protiens in normal and disease states. Because this information need is very broad but at the same time centered around the topic genes, we use only the gene names to form keyword queries. We do not use the names of the organisms because our preliminary experiment results show that including the organism names may hurt the performance. For the gene in each topic, several types of names are given, including the official name, the official symbol, the alias symbols, the product, etc. These types of names fall into two categories: the gene names and gene products are usually long, descriptive names, such as *chemokine (C-C motif) ligand 3*, and the gene symbols are short, symbolic names, such as *CCL3* and *MIP1A*. We thus form two groups of keyword queries from the 2003 topics. For each 2003 topic, we use the union of the topic gene's descriptive names to form a *name query*, and we use the union of the gene's symbolic names to form a *symbol query*. In the end, we get 50 keyword name queries and 50 keyword symbol queries from the 2003 topics. This separation presumably captures two possible types of real-world queries from biology researchers.

The 50 topics from TREC 2004 each consist of a title field, an information need field, and a context field. These topics may or may not contain a gene or protein name. Our preliminary experiment results show that using only the information need field to form queries gives the best retrieval performance. We thus use the text in the information need field only to form 50 verbose queries. These queries often contain common English words as background words, such as *about* in the query "Find articles about Ferroportin-1, an iron transporter, in humans."

The 50 topics from TREC 2005 are structured topics with templates. There are 5 templates representing 5 kinds of information need. For example, one template is "Provide information about the role of the gene X involved in the disease Y." We exclude those template background words such as *provide* and *information* when forming the queries. After removing the background words, most queries still contain more than 5 words, including some English words, so we still consider them verbose queries. We further divide the queries into two groups: queries that involve at least one gene (queries belonging to Templates 2, 3, 4 and 5), and queries that do not involve any gene (queries belonging to Template 1). We exclude Topic 135 because it does not have any judged relevant document. We thus get 39 *gene queries* and 10 *non-gene queries* from the 2005 topics.

To summarize, we use 5 sets of queries for evaluation: 50 keyword gene symbol queries from TREC 2003, 50 keyword gene name queries from TREC 2003, 50 verbose mixed queries from TREC 2004, 39 verbose gene queries from TREC 2005, and 10 verbose non-gene queries from TREC 2005. The 2004 queries are more verbose than the 2005 queries. We do not consider query expansion because the goal of our study is not to improve the absolute retrieval performance, but rather to compare the tokenization strategies.

5.2 Tokenization Heuristics

To evaluate the tokenization heuristics, we first compare the retrieval performance before and after the removal of non-functional characters. Concluding that removing the non-functional characters is safe, we then further apply the three break point normalization methods in conjunction with the three sets of break points. This gives us 9 sets of experiments. We then evaluate the Greek alphabet normalization heuristic by applying it on top of each break point normalization method in conjunction with the best set of break points. All experiments are run on each set of queries.

5.2.1 Removal of Non-Functional Characters

Table 4 shows the comparison of the retrieval performance between the naive tokenization method and the tokenization method that removes the non-functional characters. We refer to the latter method as the baseline method because it is applicable to general English text as well. The 3rd row shows the relative improvement brought by the baseline

[1]http://www.lemurproject.org/
[2]http://ir.ohsu.edu/genomics/

Table 5: Comparison Among Three Sets of Break Points

		Keyword Queries				Verbose Queries					
		03 Symbol		03 Name		04		05 Gene		05 Non-gene	
		KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF
H-Norm	BP1	0.1548	**0.1642**	0.0890	**0.0978**	0.2695	0.2663	0.2431	0.2421	**0.1569**	**0.1415**
	BP2	0.1530	0.1625	**0.0976**	0.0940	**0.2702**	**0.2676**	0.2431	0.2421	**0.1569**	**0.1415**
	BP3	**0.1598**	0.1614	0.0872	0.0904	0.2678	0.2671	**0.2447**	**0.2442**	0.1567	0.1414
S-Norm	BP1	**0.1466**	**0.1512**	0.1056	**0.1070**	**0.3015**	**0.3202**	**0.2807**	**0.2756**	**0.1836**	**0.1664**
	BP2	0.1434	0.1465	**0.1078**	0.1045	0.2986	0.3122	0.2793	0.2745	0.1835	0.1650
	BP3	0.1276	0.1016	0.0969	0.0900	0.2406	0.2339	0.2757	0.2578	0.1795	0.1654
J-Norm	BP1	**0.1751**	**0.1750**	**0.0874**	**0.0930**	0.2742	0.2824	0.2474	**0.2469**	**0.1571**	**0.1417**
	BP2	0.1738	0.1735	0.0869	0.0894	**0.2749**	**0.2838**	**0.2475**	**0.2469**	0.1569	**0.1417**
	BP3	0.1738	0.1735	0.0869	0.0894	**0.2749**	**0.2838**	**0.2475**	**0.2469**	0.1569	**0.1417**

Table 6: Comparison Among Break Point Normalization Methods

		Keyword Queries				Verbose Queries					
		03 Symbol		03 Name		04		05 Gene		05 Non-gene	
		KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF
BP1	H-Norm	0.1548	0.1642	0.0890	0.0978	0.2695	0.2663	0.2431	0.2421	0.1569	0.1415
	S-Norm	0.1466	0.1512	**0.1056**	**0.1070**	**0.3015**	**0.3202**	**0.2807**	**0.2756**	**0.1836**	**0.1664**
	J-Norm	**0.1751**	**0.1750**	0.0874	0.0930	0.2742	0.2824	0.2474	0.2469	0.1571	0.1417
BP2	H-Norm	0.1530	0.1625	0.0976	0.0940	0.2702	0.2676	0.2431	0.2421	0.1569	0.1415
	S-Norm	0.1434	0.1465	**0.1078**	**0.1045**	**0.2986**	**0.3122**	**0.2793**	**0.2745**	**0.1835**	**0.1650**
	J-Norm	**0.1738**	**0.1735**	0.0869	0.0894	0.2749	0.2838	0.2475	0.2469	0.1569	0.1417
BP3	H-Norm	0.1598	0.1614	0.0872	**0.0904**	0.2678	0.2671	0.2447	0.2442	0.1567	0.1414
	S-Norm	0.1276	0.1016	**0.0969**	0.0900	0.2406	0.2339	**0.2757**	**0.2578**	**0.1795**	**0.1654**
	J-Norm	**0.1738**	**0.1735**	0.0869	0.0894	**0.2749**	**0.2838**	0.2475	0.2469	0.1569	0.1417

method. The 4th row shows the number of queries whose baseline MAP measure is lower than the naive MAP measure, out of the total number of queries in each query set. We can see that in all sets of queries, the baseline method outperforms the naive method. The improvement is especially substantial with the verbose queries. The explanation is probably that there are more English words in the verbose queries than in the keyword queries. A tokenization method that separates words from punctuation can make the term frequencies and inverted document frequencies more accurate, especially for English words. Therefore, baseline method has more impact on verbose queries.

It is shown in Table 4 that for each query set, there are still some queries whose baseline MAP measure is lower than the naive MAP measure. We randomly picked some of these topics and did an error analysis. We found that in most cases the performance drop was not caused by bad tokenization, but rather by some information need that was not captured in the queries. We can thus still conclude that it is in general safe to remove those non-functional characters.

5.2.2 Break Points

Table 5 shows the comparison among the three sets of break points when one of H-Norm, S-Norm and J-Norm is used in conjunction. The best performance among the three sets for each set of queries is shown in bold font. The pattern is not consistent, but we can see that BP1 performs the best for the most number of experiments, especially when S-Norm is used, or when J-Norm is used for the keyword queries. As we will show next, S-Norm and J-Norm are preferred over H-Norm. Thus we can conclude that BP1 is the best set of break points to use.

5.2.3 Break Point Normalization

Table 6 shows the comparison among the three break point normalization methods when each set of break points is used in conjunction. It is very clear from Table 6, especially from the three rows with BP1 and the three rows with BP2, that for the symbol queries, J-Norm performs the best, and for the name queries and the verbose queries, S-Norm performs the best. This suggests that for gene symbols that are combinations of alphabetical characters, numerical characters and special characters such as hyphens, removing the special characters is the most effective way to normalize different lexical variants of the same name. For keyword name queries and verbose queries, however, most of the query words are not gene symbols. Replacing special characters such as hyphens and slashes with spaces is the most effective normalization method. This is probably not only because S-Norm may help normalize the gene names but also because it effectively separates hyphenated compound words such as *CCAAT/enhancer-binding* and *azaserine-induced*, which are better to be separated for retrieval purpose.

5.2.4 Greek Alphabet Normalization

In Table 7, we show the effect of applying G-Norm on top H-Norm, S-Norm and J-Norm when the best set of break points, BP1, is used. Although the pattern is not very clear in the comparison, we can see that in most cases, Greek

alphabet normalization does not improve the performance. However, for keyword symbol queries, when J-Norm is used, G-Norm does improve the performance a little bit. Since J-Norm is the best break point normalization method for keyword symbol queries, we can conclude that we should apply G-Norm on top of J-Norm when the queries are keyword symbol queries.

5.2.5 Improvement Summary

From the above comparisons, we can draw the following conclusions. First, we can safely remove those non-functional characters as defined in Section 3.1. Second, BP1 is the best set of break points to use for break point normalization. Third, for keyword symbol queries, J-Norm is the most effective break point normalization method, and for keyword name queries and verbose queries, S-Norm is the most effective normalization method. Fourth, Greek alphabet normalization in general is not effective except when J-Norm is used for the keyword symbol queries.

In Table 8, we show the relative improvement brought by each tokenization heuristic and the overall improvement over the naive method. Except for the overall improvement, the percentage of improvement shown in the table is the improvement with respect to the previous row. The keyword name queries from 2003 and the verbose queries are grouped into a single category (the non-symbol queries) because the same set of heuristics work the best on them. The gene queries and the nongene queries from 2005 are also combined. We can see from the table that when a set of suitable tokenization heuristics are used for each type of queries, the performance can improve by at least 15% for all sets of queries. The improvement is mostly substantial for the 2004 and 2005 queries, which are verbose queries. For 2004 queries, the improvement is mostly brought by removal of the non-functional characters. The reason may be that 2004 queries contain more background English words than the 2005 queries. It therefore suggests that the more verbose a query is, i.e., the more background English words a query contains, the more important it is to carefully remove the non-functional characters when doing tokenization. For 2003 name queries and 2005 queries, however, the break point normalization step contributes more than removal of the non-functional characters to the final improvement. This suggests that for queries containing gene names, normalization of break points is important in tokenization.

5.3 Stemming and Stop Word Removal

5.3.1 Stemming

In this section, we compare non-stemming and stemming performance, and compare the different stemming algorithms. We apply three stemmers, the Porter stemmer, the Lovins stemmer, and the S stemmer, on top of the best tokenization strategy for each set of queries. The performance is shown in Table 9. We can see that for keyword queries, all stemmers decrease the performance. However, for verbose queries, in most cases all three stemmers improve the performance. Porter stemmer and Lovins stemmer are shown to be better than the S stemmer. The comparison between the Porter stemmer and the Lovins stemmer is mixed.

5.3.2 Stop Word Removal

In this section, we compare the two stop word removal methods. Method one uses the PubMed stop word list, which consists of 132 common English words. Method two uses a collection-based stop word list, i.e., the most frequent k words in the same document collection for retrieval. We use different values of k to see how this cutoff number affects the performance. Table 10 shows the performance on each query set before and after stop word removal. The best performance in each column is shown in bold font. We can see that except for 2004 queries, stop word removal either does not improve the performance, or only slightly improves the performance when a few number of collection-based stop words are removed. For 2004 queries, however, using the external stop word list is better than using the collection based stop words. This may be because of some common English words in 2004 queries that are not captured by the collection based stop word list. For example, the word *about* in the query "Find articles about Ferroportin-1, an iron transporter, in humans." is in the PubMed stop word list, but is only the 434th frequent word in the document collection.

6. CONCLUSIONS

Because of the irregular forms of entity names and their lexical variants in the biomedical text, appropriate tokenization is an important preprocessing step in biomedical IR. In this paper, we systematically evaluated a set of tokenization strategies generalized from existing work, including a non-functional character removal step, a break point normalization step with three possible normalization methods and three possible sets of break points, and a Greek alphabet normalization step. We also empirically studied the effect of stemming and stop word removal for biomedical IR. Our evaluation was conducted on all the available TREC biomedical IR test collections, and we employed two representative retrieval methods. Results from both retrieval methods show that tokenization can significantly affect the retrieval performance, as we expected; appropriate tokenization can improve the retrieval performance by up to 80%.

Our findings provide guidelines on how to do tokenization for IR and text mining on biomedical text. In particular, the general recommendations are: First, non-functional characters should be removed from the text using a set of heuristic rules. Second, for different types of queries, different tokenization heuristics should be applied. For queries that contain only gene symbols, removing brackets, hyphens, slashes and underlines in the tokens and replacing Greek letters with their Latin equivalents are useful. For queries that contain only full gene names and for verbose queries that also contain English words, replacing brackets, hyphens, slashes and underlines with spaces should be used. Numerical characters should not be separated from alphabetical characters. Third, for verbose queries, Porter or Lovins stemmer can be used to further improve the performance. Finally, stop word removal should not be performed except for verbose queries containing common English words, in which case an external stop word list may help.

7. REFERENCES

[1] R. K. Ando, M. Dredze, and T. Zhang. TREC 2005 genomics track experiments at IBM Watson. In *TREC-2005*.

Table 7: The Effect of Greek Alphabet Normalization

		Keyword Queries				Verbose Queries					
		03 Symbol		03 Name		04		05 Gene		05 Non-gene	
	G-Norm	KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF
H-Norm	No	**0.1548**	0.1642	**0.0890**	**0.0978**	**0.2695**	**0.2663**	**0.2431**	**0.2421**	**0.1569**	**0.1415**
	Yes	0.1541	**0.1649**	0.0870	0.0952	0.2674	0.2646	0.2426	0.2415	0.1563	**0.1415**
S-Norm	No	**0.1466**	0.1512	**0.1056**	**0.1070**	**0.3015**	**0.3202**	0.2807	**0.2756**	**0.1836**	**0.1664**
	Yes	0.1452	**0.1516**	0.1034	0.1033	0.2997	0.3186	**0.2812**	0.2754	**0.1836**	0.1663
J-Norm	No	0.1751	0.1750	**0.0874**	**0.0930**	**0.2742**	0.2824	**0.2474**	**0.2469**	**0.1571**	**0.1417**
	Yes	**0.1784**	**0.1780**	0.0854	0.0908	0.2741	**0.2833**	0.2435	0.2415	0.1565	**0.1417**

Table 8: The Relative Improvement Brought by Each Tokenization Heuristic

Keyword Symbol Queries			Non-Symbol Queries						
	03			03		04		05	
	KL	TFIDF		KL	TFIDF	KL	TFIDF	KL	TFIDF
Naive	0.1451	0.1546	Naive	0.0833	0.0919	0.1672	0.1736	0.1795	0.1773
Baseline	0.1520	0.1643	Baseline	0.0892	0.0971	0.2694	0.2700	0.2143	0.2031
% Impr.	**4.76%**	**6.27%**	**% Impr.**	**7.08%**	**5.66%**	**61.1%**	**55.5%**	**19.4%**	**14.6%**
BP1+J-Norm	0.1751	0.1750	BP1+S-Norm	0.1056	0.1070	0.3015	0.3202	0.2608	0.2533
% Impr.	**15.2%**	**5.61%**	**% Impr.**	**18.4%**	**10.2%**	**11.9%**	**18.6%**	**21.7%**	**24.7%**
+G-Norm	0.1784	0.1780							
% Impr.	**1.88%**	**1.71%**							
Overall % Impr. over Naive	**23.0%**	**15.1%**	**Overall % Impr. over Naive**	**26.8%**	**16.4%**	**80.3%**	**84.5%**	**45.3%**	**42.9%**

[2] S. Buttcher, C. L. A. Clarke, and G. V. Cormack. Domain-specific synonym expansion and validation for biomedical information retrieval. In *TREC-2004*.

[3] B. Carpenter. Phrasal queries with LingPipe and Lucene: Ad hoc genomics text retrieval. In *TREC-2004*.

[4] C. Crangle, A. Zbyslaw, J. M. Cherry, and E. L. Hong. Concept extraction and synonym management for biomedical information retrieval. In *TREC-2004*.

[5] A. Dayanik, C. G. Nevill-Manning, and R. Oughtred. Partitioning a graph of sequences, structures and abstracts for information retrieval. In *TREC-2003*.

[6] S. Fujita. Revisiting again document length hypotheses TREC-2004 genomics track experiments at patolis. In *TREC-2004*.

[7] D. Harman. How effective is suffixing? *Journal of the American Society for Information Science*, 42(1):7–15, 1991.

[8] X. Huang, M. Zhong, and L. Si. York University at TREC 2005: Genomics track. In *TREC-2005*.

[9] J. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31, 1968.

[10] A. Pirkola and E. Leppanen. TREC 2003 genomics track experiments at UTA. In *TREC-2003*.

[11] M. Porter. An algorithm for suffix stripping. In *Readings in Information Retrieval*, pages 313–316. Morgan Kaufmann Publishers Inc.

[12] J. Savoy, Y. Rasolofo, and L. Perret. Report on the TREC 2003 experiment. In *TREC-2003*.

[13] Y.-I. Song, K.-S. Han, H.-C. Seo, S.-B. Kim, and H.-C. Rim. Biomedical text retrieval system at Korea University. In *TREC-2003*.

[14] S. Tomlinson. Robust, web and genomics retrieval with Hummingbird SearchServer at TREC 2003. In *TREC-2003*.

[15] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *CIKM-2001*.

Table 9: The Effect of Stemming

	Keyword Queries				Verbose Queries					
	03 Symbol		03 Name		04		05 Gene		05 Non-gene	
	KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF
Best Tokenization	**0.1784**	**0.1780**	**0.1056**	**0.1070**	0.3015	0.3202	0.2807	0.2756	0.1836	0.1664
+Porter	0.1751	0.1741	0.1052	0.1023	**0.3380**	**0.3329**	0.2901	0.2872	**0.2094**	0.1601
+Lovins	0.1726	0.1755	0.1054	0.1016	0.3130	0.3165	**0.2966**	**0.2965**	0.2036	**0.1755**
+S	0.1757	0.1747	0.1045	0.1028	0.3234	0.3297	0.2840	0.2850	0.1962	0.1683

Table 10: The Effect of Stop Word Removal

Stop Word List	k	Keyword Queries				Verbose Queries					
		03 Symbol		03 Name		04		05 Gene		05 Non-gene	
		KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF	KL	TFIDF
No	0	0.1784	**0.1780**	0.1056	0.1070	0.3380	0.3329	0.2901	**0.2874**	0.2094	0.1601
PubMed	132	0.1772	0.1780	0.1028	**0.1071**	**0.3487**	**0.3366**	0.2916	0.2846	0.1962	0.1581
Collection Based	5	0.1777	0.1780	**0.1061**	0.1070	0.3389	0.3330	**0.2936**	0.2873	**0.2098**	0.1602
	10	**0.1786**	0.1780	0.1047	0.1070	0.3393	0.3330	0.2925	0.2873	0.1964	**0.1604**
	20	0.1777	0.1780	0.1046	0.1070	0.3374	0.3317	0.2921	0.2829	0.1768	0.1508
	100	0.1769	0.1780	0.0872	0.0990	0.3264	0.3242	0.2770	0.2744	0.1533	0.1393